DL-1-Performing matrix multiplication and finding eigen vectors and eigen values using TensorFlow.ipynb

```
import tensorflow as tf

print("Matrix Multiplication Demo")
x = tf.constant([1, 2, 3, 4, 5, 6], shape=[2, 3])
print(x)

y = tf.constant([7, 8, 9, 10, 11, 12], shape=[3, 2])
print(y)

z = tf.matmul(x, y)
print("Product:", z)

e_matrix_A = tf.random.uniform(
   [2, 2], minval=3, maxval=10, dtype=tf.float32, name="matrixA"
)
print("Matrix A:\n{}\n\n".format(e_matrix_A))
eigen_values_A, eigen_vectors_A = tf.linalg.eigh(e_matrix_A)
print(
   "Eigen Vectors:\n{}\n\nEigen Values:\n{}\n".format(eigen_vectors_A, eigen_values_A)
)
```

*********** *********** *********** *********** *********** *********** ***********

#DL-2-Solving XOR problem using deep feed forward network.ipynb

```
import numpy as np
from keras.layers import Dense
from keras.models import Sequential

# Create a sequential model
model = Sequential()

# Add layers to the model
model.add(Dense(units=2, activation='relu', input_dim=2))
model.add(Dense(units=1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Print model summary
print(model.summary())

# Print initial weights
```

```
print("Initial weights:")
print(model.get_weights())

# Define the input data and labels
X = np.array([[0., 0.], [0., 1.], [1., 0.], [1., 1.]])
Y = np.array([0., 1., 1., 0.])

# Fit the model
model.fit(X, Y, epochs=1000, batch_size=4, verbose=1)

# Print weights after training
print("Weights after training:")
print(model.get_weights())

# Make predictions
print("Predictions:")
print(model.predict(X, batch_size=4))
```

*********** *********** *********** *********** *********** *********** ***********

```
# Aim: Implementing deep neural network for performing binary classification task.
# pip install keras
from keras.models import Sequential
from keras.layers import Dense
import pandas as pd

names = [
  "No. of pregnancies",
  "Glucose level",
  "Blood Pressure",
  "skin thickness",
  "Insulin",
  "BMI",
  "Diabetes pedigree",
  "Age",
  "Class",
]

#csv file with no column names expected
df = pd.read_csv("/content/pima-indians-diabetes.data.csv", names=names)
df.head(3)
binaryc = Sequential()

from tensorflow.tools.docs.doc_controls import doc_in_current_and_subclasses

binaryc.add(Dense(units=10, activation="relu", input_dim=8))
binaryc.add(Dense(units=8, activation="relu"))
```

```
binaryc.add(Dense(units=1, activation="sigmoid"))
binaryc.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size=0.25, random_state=1)
xtrain.shape
ytrain.shape
binaryc.fit(xtrain, ytrain, epochs=200, batch_size=20)
predictions = binaryc.predict(xtest)
predictions.shape
class_labels = []
for i in predictions:
    if i > 0.5:
        class_labels.append(1)
    else:
        class_labels.append(0)
class_labels
from sklearn.metrics import accuracy_score

print("Accuracy Score", accuracy_score(ytest, class_labels))
```

*********** *********** *********** *********** *********** *********** ***********

```
#DL-4a-Using feed Forward Network with multiple hidden layers for performing multiclass classification and predicting the class.ipynb
#------required flower_1.csv DATA SET

from keras.models import Sequential
from keras.layers import Dense
import pandas as pd
import numpy as np

df = pd.read_csv("/content/flower_1.csv")

#df = pd.read_csv("data/flower_1.csv")
# df = pd.read_csv("flower_1.csv")

df.head()

x=df.iloc[:,:-1].astype(float)
y=df.iloc[:,-1]

print(x.shape)
print(y.shape)
```

```
#labelencode y
from sklearn.preprocessing import LabelEncoder
lb=LabelEncoder()
y=lb.fit_transform(y)
y

import numpy as np
from tensorflow.keras.utils import to_categorical
#from keras.utils import np_utils
encoded_Y = to_categorical(y)
encoded_Y

#creating a model
model = Sequential()

model.add(Dense(units = 10, activation = 'relu', input_dim = 4))
model.add(Dense(units = 8, activation = 'relu'))
model.add(Dense(units = 3, activation = 'softmax'))

model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])

model.fit(x,encoded_Y,epochs = 400,batch_size = 10)

predict = model.predict(x)
print(predict)

for i in range(35,150,3):
   print(predict[i],encoded_Y[i])

actual = []

for i in range(0,150):
   actual.append(np.argmax(predict[i]))

print(actual)

newdf = pd.DataFrame(list(zip(actual,y)),columns = ['Actual','Predicted'])
newdf
```

*********** *********** *********** *********** *********** *********** ***********

```
#DL-4b-Using a deep feed forward network with two hidden layers for performing classification and predicting the probability of class.ipynb

import tensorflow as tf
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification

# Create a synthetic dataset for binary classification
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Convert the labels to one-hot encoded format for categorical crossentropy loss
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Initialize the model
model = Sequential()

# Add the first hidden layer with 64 neurons and ReLU activation function
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))

# Add the second hidden layer with 32 neurons and ReLU activation function
model.add(Dense(32, activation='relu'))

# Add the output layer with softmax activation for classification (2 classes)
model.add(Dense(2, activation='softmax'))

# Compile the model with categorical crossentropy loss, Adam optimizer, and accuracy as a metric
model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])

# Train the model with the training data
model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model on the test data
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy*100:.2f}%")

# Predict class probabilities for the test data
probabilities = model.predict(X_test)

# Display the first 5 predictions
print(probabilities[:5])
```

*********** *********** *********** *********** *********** *********** ***********

#DL-4c-Using a deep feed forward network with two hidden layers for performing linear regression and predicting values.ipynb

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_regression

# Create a synthetic dataset for regression
X, y = make_regression(n_samples=1000, n_features=20, noise=0.1, random_state=42)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the model
model = Sequential()

# Add the first hidden layer with 64 neurons and ReLU activation function
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))

# Add the second hidden layer with 32 neurons and ReLU activation function
model.add(Dense(32, activation='relu'))

# Add the output layer with no activation function (linear output for regression)
model.add(Dense(1))

# Compile the model with mean squared error loss, Adam optimizer, and mean absolute error as a
metric
model.compile(loss='mean_squared_error', optimizer=Adam(), metrics=['mean_absolute_error'])

# Train the model with the training data
model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model on the test data
loss, mae = model.evaluate(X_test, y_test)
print(f"Test Mean Absolute Error: {mae:.2f}")

# Predict values for the test data
predictions = model.predict(X_test)

# Display the first 5 predictions and actual values
print("Predictions:", predictions[:5].flatten())
print("Actual values:", y_test[:5])
```

*********** *********** *********** *********** *********** *********** ***********

#DL-5a-Evaluating feed forward deep network for regression using KFold cross validation.ipynb

```
!pip install keras (2.15.0)
!pip install scikit_learn
!pip install scikeras

import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
# from keras.wrappers.scikit_learn import KerasRegressor
from scikeras.wrappers import KerasRegressor
from sklearn.model_selection import cross_val_score, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.neural_network import MLPRegressor


#dataframe = pd.read_csv("MscIT\Semester 4\Deep_Learning\Practical05\housing.csv")
dataframe = pd.read_csv("/content/housing.csv")
dataset = dataframe.values

# Print the shape of dataset to verify the number of features and samples
print("Shape of dataset:", dataset.shape)

# Ensure correct slicing for features and target variable
X = dataset[:, :-1]  # Select all columns except the last one as features
Y = dataset[:, -1]   # Select the last column as target variable

def wider_model():
    model = Sequential()
    model.add(Dense(15, input_dim=13, kernel_initializer='normal', activation='relu'))
    # model.add(Dense(20, input_dim=13, kernel_initializer='normal', activation='relu'))
    model.add(Dense(13, kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, kernel_initializer='normal'))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(build_fn=wider_model, epochs=10, batch_size=5)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)

results = cross_val_score(pipeline, X, Y, cv=kfold)
print("Wider: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

*********** *********** *********** *********** *********** *********** ***********

5B. Evaluating feed forward deep network for multiclass Classification using KFold cross-validation.

```
!pip install scikeras
!pip install np_utils

# loading libraries
import pandas
from keras.models import Sequential
from keras.layers import Dense
from scikeras.wrappers import KerasClassifier
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder

# loading dataset
df = pandas.read_csv('/content/flowers.csv', header=0)
print(df)

# splitting dataset into input and output variables
X = df.iloc[:, 0:4].astype(float)
y = df.iloc[:, 4]
# print(X)
# print(y)

# encoding string output into numeric output
encoder = LabelEncoder()
encoder.fit(y)
encoded_y = encoder.transform(y)
print(encoded_y)
dummy_Y = to_categorical(encoded_y)
print(dummy_Y)

def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(8, input_dim=4, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

```
estimator = baseline_model()
estimator.fit(X, dummy_Y, epochs=100, shuffle=True)
action = estimator.predict(X)
for i in range(25):
   print(dummy_Y[i])
   print('^^^^^^^^^^^^^^^^^^^^^^')
for i in range(25):
   print(action[i])
```

*********** *********** *********** *********** *********** *********** ***********

#DL-6a-Implementing regularization to avoid overfitting in binary classification.ipynb

```
from matplotlib import pyplot
from sklearn.datasets import make_moons
from keras.models import Sequential
from keras.layers import Dense
X,Y=make_moons(n_samples=100,noise=0.2,random_state=1)
n_train=30
trainX,testX=X[:n_train,:],X[n_train:]
trainY,testY=Y[:n_train],Y[n_train:]
#print(trainX)
#print(trainY)
#print(testX)
#print(testY)
model=Sequential()
model.add(Dense(500,input_dim=2,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
history=model.fit(trainX,trainY,validation_data=(testX,testY),epochs=1000)
pyplot.plot(history.history['accuracy'],label='train')
pyplot.plot(history.history['val_accuracy'],label='test')
pyplot.legend()
pyplot.show()

pyplot.plot(history.history['accuracy'],label='train')
pyplot.plot(history.history['val_accuracy'],label='test')
pyplot.legend()
pyplot.show()
```

*********** *********** *********** *********** *********** *********** ***********

#DL-6b-Implement l2 regularization with alpha=0.001.ipynb

```
from matplotlib import pyplot
from sklearn.datasets import make_moons
from keras.models import Sequential
from keras.layers import Dense
from keras.regularizers import l2

X,Y=make_moons(n_samples=100,noise=0.2,random_state=1)
n_train=30
trainX,testX=X[:n_train,:],X[n_train:]
trainY,testY=Y[:n_train],Y[n_train:]
#print(trainX)
#print(trainY)
#print(testX)
#print(testY)
model=Sequential()
model.add(Dense(500,input_dim=2,activation='relu',kernel_regularizer=l2(0.001)))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
history=model.fit(trainX,trainY,validation_data=(testX,testY),epochs=1000)

pyplot.plot(history.history['accuracy'],label='train')
pyplot.plot(history.history['val_accuracy'],label='test')
pyplot.legend()
pyplot.show()
```

*********** *********** *********** *********** *********** *********** ***********

```
#DL-6c-Replace l2 regularization with l2 regularization.ipynb

# !pip install pandas
# !pip install matplotlib
# !pip install keras
# !pip install tensorflow

from matplotlib import pyplot
from sklearn.datasets import make_moons
from keras.models import Sequential
from keras.layers import Dense
from keras.regularizers import l1_l2
X,Y=make_moons(n_samples=100,noise=0.2,random_state=1)
n_train=30
trainX,testX=X[:n_train,:],X[n_train:]
trainY,testY=Y[:n_train],Y[n_train:]
#print(trainX)
#print(trainY)
#print(testX)
```

```
#print(testY)

model=Sequential()
model.add(Dense(500,input_dim=2,activation='relu',kernel_regularizer=l1_l2(l1=0.001,l2=0.001)))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
history=model.fit(trainX,trainY,validation_data=(testX,testY),epochs=400)
pyplot.plot(history.history['accuracy'],label='train')
pyplot.plot(history.history['val_accuracy'],label='test')
pyplot.legend()
pyplot.show()
```

*********** *********** *********** *********** *********** *********** ***********

#DL-7-Demonstrate recurrent neural network that learns to perform sequence analysis for stock price.ipynb

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout
from sklearn.preprocessing import MinMaxScaler

# Read training dataset
dataset_train = pd.read_csv('/content/Google_Stock_Price_Train.csv')
training_set = dataset_train.iloc[:, 1:2].values

# Scale the training set
sc = MinMaxScaler(feature_range=(0,1))
training_set_scaled = sc.fit_transform(training_set)

# Create X_train and Y_train
X_train = []
Y_train = []
for i in range(60, 1258):
   X_train.append(training_set_scaled[i-60:i, 0])
   Y_train.append(training_set_scaled[i, 0])
X_train, Y_train = np.array(X_train), np.array(Y_train)

# Reshape X_train for LSTM
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

# Build the LSTM model
regressor = Sequential()
regressor.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
```

```
regressor.add(Dropout(0.2))
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units=50, return_sequences=True))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units=50))
regressor.add(Dropout(0.2))
regressor.add(Dense(units=1))
regressor.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
regressor.fit(X_train, Y_train, epochs=100, batch_size=32)

# Read test dataset
dataset_test = pd.read_csv('/content/Google_Stock_Price_Test.csv')
real_stock_price = dataset_test.iloc[:, 1:2].values

# Concatenate total dataset
dataset_total = pd.concat((dataset_train['Open'], dataset_test['Open']), axis=0)
inputs = dataset_total[len(dataset_total)-len(dataset_test)-60:].values
inputs = inputs.reshape(-1, 1)
inputs = sc.transform(inputs)

# Create X_test
X_test = []
for i in range(60, 80):
    X_test.append(inputs[i-60:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

# Predict stock prices
predicted_stock_price = regressor.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)

# Visualize results
plt.plot(real_stock_price, color='red', label='Real Google Stock Price')
plt.plot(predicted_stock_price, color='blue', label='Predicted Stock Price')
plt.xlabel('Time')
plt.ylabel('Google Stock Price')
plt.legend()
plt.show()
```

*********** *********** *********** *********** *********** *********** ***********

8. Performing encoding and decoding of images using deep autoencoder.

```
import keras
from keras import layers
from keras.datasets import mnist
import numpy as np

encoding_dim = 32

# this is our input image
input_img = keras.Input(shape=(784,))

# "encoded" is the encoded representation of the input
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)

# "decoded" is the lossy reconstruction of the input
decoded = layers.Dense(784, activation='sigmoid')(encoded)

# creating autoencoder model
autoencoder = keras.Model(input_img, decoded)

# create the encoder model
encoder = keras.Model(input_img, encoded)

encoded_input = keras.Input(shape=(encoding_dim,))

# Retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]

# create the decoder model
decoder = keras.Model(encoded_input, decoder_layer(encoded_input))

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# scale and make train and test dataset
(X_train, _), (X_test, _) = mnist.load_data()
X_train = X_train.astype('float32') / 255.
X_test = X_test.astype('float32') / 255.
X_train = X_train.reshape((len(X_train), np.prod(X_train.shape[1:])))
X_test = X_test.reshape((len(X_test), np.prod(X_test.shape[1:])))

print(X_train.shape)
print(X_test.shape)

# train autoencoder with training dataset
autoencoder.fit(X_train, X_train,
        epochs=50,
        batch_size=256,
        shuffle=True,
```

```
        validation_data=(X_test, X_test))

encoded_imgs = encoder.predict(X_test)
decoded_imgs = decoder.predict(encoded_imgs)

import matplotlib.pyplot as plt

n = 10  # How many digits we will display
plt.figure(figsize=(40, 4))
for i in range(10):
    # display original
    ax = plt.subplot(3, 20, i + 1)
    plt.imshow(X_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display encoded image
    ax = plt.subplot(3, 20, i + 1 + 20)
    plt.imshow(encoded_imgs[i].reshape(8, 4))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstruction
    ax = plt.subplot(3, 20, 2 * 20 + i + 1)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()
```

*********** *********** *********** *********** *********** *********** ***********

```
# 9. Aim: Implementation of convolutional neural network to predict number from number images

from keras.datasets import mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten
import matplotlib.pyplot as plt

# Download MNIST data and split into train and test sets
```

```
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()

# Plot the first image in the dataset
plt.imshow(X_train[0])
plt.show()
print(X_train[0].shape)

# Reshape data for CNN (add channel dimension)
X_train = X_train.reshape(60000, 28, 28, 1)
X_test = X_test.reshape(10000, 28, 28, 1)

# One-hot encode labels
Y_train = to_categorical(Y_train)
Y_test = to_categorical(Y_test)

# Print an example of one-hot encoded label
print(Y_train[0])

# Define the model architecture
model = Sequential()

# Learn image features with convolutional layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28, 28, 1)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Flatten())

# Add a dense layer with softmax activation for 10-class classification
model.add(Dense(10, activation='softmax'))

# Compile the model for training
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model with validation data
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=3)

# Make predictions on the first 4 test images
predictions = model.predict(X_test[:4])
print(predictions)  # Predicted probabilities for each class

# Print the actual labels for the first 4 test images
print(Y_test[:4])  # One-hot encoded labels
```

*********** *********** *********** *********** *********** *********** ***********

10. Denoising of images using autoencoder.

```
import keras
from keras.datasets import mnist
from keras import layers
import numpy as np
from keras.callbacks import TensorBoard
import matplotlib.pyplot as plt

(X_train,_),(X_test,_)=mnist.load_data()
X_train=X_train.astype('float32')/255.
X_test=X_test.astype('float32')/255.
X_train=np.reshape(X_train,(len(X_train),28,28,1))
X_test=np.reshape(X_test,(len(X_test),28,28,1))
noise_factor=0.5
X_train_noisy=X_train+noise_factor*np.random.normal(loc=0.0,scale=1.0,size=X_train.shape)
X_test_noisy=X_test+noise_factor*np.random.normal(loc=0.0,scale=1.0,size=X_test.shape)
X_train_noisy=np.clip(X_train_noisy,0.,1.)
X_test_noisy=np.clip(X_test_noisy,0.,1.)

n=10
plt.figure(figsize=(20,2))
for i in range(1,n+1):
   ax=plt.subplot(1,n,i)
   plt.imshow(X_test_noisy[i].reshape(28,28))
   plt.gray()
   ax.get_xaxis().set_visible(False)
   ax.get_yaxis().set_visible(False)
plt.show()

input_img=keras.Input(shape=(28,28,1))
x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(input_img)
x=layers.MaxPooling2D((2,2),padding='same')(x)
x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(x)
encoded=layers.MaxPooling2D((2,2),padding='same')(x)
x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(encoded)
x=layers.UpSampling2D((2,2))(x)
x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(x)
x=layers.UpSampling2D((2,2))(x)
decoded=layers.Conv2D(1,(3,3),activation='sigmoid',padding='same')(x)

autoencoder=keras.Model(input_img,decoded)
autoencoder.compile(optimizer='adam',loss='binary_crossentropy')
autoencoder.fit(X_train_noisy,X_train, epochs=3, batch_size=128, shuffle=True,
validation_data=(X_test_noisy,X_test),
callbacks=[TensorBoard(log_dir='/tmo/tb',histogram_freq=0,write_graph=False)])

predictions=autoencoder.predict(X_test_noisy)
```

```
m=10
plt.figure(figsize=(20,2))
for i in range(1,m+1):
    ax=plt.subplot(1,m,i)
    plt.imshow(predictions[i].reshape(28,28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```